2Ш

УДК 004.051:004.89:519.712.2

В.И. Шинкаренко, Г.Г. Кроль, И.В. Литвин, Е.Г. Васецкий
Днепропетровский национальный университет железнодорожного транспорта
имени академика В. Лазаряна, г. Днепропетровск, Украина
ccp@diit-70.dp.ua

Методы и средства структурной адаптации алгоритмов на метаалгоритмической основе

Рассматривается задача структурной адаптации алгоритмов. Представлены достаточно универсальные средства адаптации алгоритмов в составе различного прикладного программного обеспечения. Разработаны полнофункциональный редактор и специализированные средства отладки метаалгоритмов, подсистема синтеза адаптивных алгоритмов. Подсистема анализа эффективности алгоритмов, основанная на кластеризации методом максиминного расстояния, вырабатывает базу знаний и тем самым управляет процессами синтеза и адаптации.

Введение

Одним из определяющих атрибутов интеллекта, в том числе и искусственного, является способность приспосабливаться к внешней среде, способность к адаптации. Адаптивные алгоритмы в современном программном обеспечении узкоспециализированные. Они применяются для решения конкретных задач и разрабатываются с учетом характерных, специфических условий их применения. Так, алгоритмы сжатия данных [1] учитывают особенности сжимаемых данных и статистическую информацию об уже обработанных данных, нейросетевые алгоритмы распознавания и принятия решений обучаются на характерной для режимов эксплуатации обучающей выборке.

Предложен подход к адаптации, при котором переход от универсальности к специализации осуществляется автоматически [2]. Основным средством этого подхода является универсальный метаалгоритм. Метаалгоритм [1] – это специальным образом заданный абстрактный алгоритм, на основе которого могут быть построены конкретные алгоритмы, обобщенный алгоритм решения некоторой задачи. Он должен включать возможности всех известных и в некоторых условиях эффективных алгоритмов решения задачи.

Вопросы моделирования метаалгоритмов в теории систем алгоритмических алгебр интегрированными алгеброалгоритмическими моделями рассмотрены в [3-5]. Нами выполнены аналогичные исследования средствами грамматико-алгоритмических структур.

Целью данной работы является разработка методов и средств, которые позволят синтезировать структурно адаптивные алгоритмы [1], [6], т.е. алгоритмы, которые при изменении среды функционирования меняют свою структуру для повышения эксплуатационных характеристик, в первую очередь временной [7] и функциональной эффективности [8]. ПО должно быть разработано так, чтобы была возможность его включения в другие прикладные системы, повышая уровень их «интеллекта».

Постановка задачи

Предложенная методика структурной адаптации алгоритмов [2] заключается в циклическом процессе:

– синтеза на основе метаалгоритма множества конкретных функционально эквивалентных алгоритмов различной структуры;

– выполнения их и измерения целевого показателя эффективности;

– отбора конкурентоспособных алгоритмов и на основе анализа подготовка рекомендаций для последующего синтеза все более эффективных алгоритмов.

Процесс адаптации считается завершенным при достижении удовлетворительных значений показателя эффективности, либо при стабилизации процесса на некотором не улучшаемом уровне эффективности.

Так как методология имеет метаалгоритмическую основу, возникает необходимость автоматизированной поддержки методов разработки и отладки метаалгоритмов.

Сложность отладки метаалгоритма заключается в том, что непосредственно выполнить алгоритм нельзя. Выполняются лишь синтезированные на его основе конкретные алгоритмы. Ошибки в метаалгоритме опосредствованно проявляются в конкретных алгоритмах. Возникает необходимость в программной среде, поддерживающей и упрощающей процесс отладки метаалгоритмов.

Второй задачей является универсализация процесса структурной адаптации алгоритма, что должно стать предпосылкой для ее применения в различных прикладных системах.

Средства разработки метаалгоритмов

Метаалгоритм строится на основе модифицированного метода пошаговой детализации: на первом шаге алгоритм записывается с помощью абстрактных операторов (АО) и предикатов первого уровня и инструкций языка программирования (ЯП); на втором шаге абстрактные операторы первого уровня расписываются (реализовываются) с помощью абстрактных операторов второго уровня и инструкций языка программирования и т.д., пока алгоритм в виде программы не будет полностью записан с помощью инструкций языка программирования.

Модификация метода заключается в том, что абстрактные операторы могут иметь несколько реализаций, каждая из которых может быть использована для синтеза конкретных алгоритмов.

Для поддержки метода пошаговой детализации разработан редактор метаалгоритмов. На рис. 1 представлена основная форма редактора. Она позволяет вести учет и контроль за последовательно разрабатываемыми реализациями АО. Форма разделена на три части: нереализованные и реализованные АО и их спецификации.

Части нереализованных и реализованных АО соответствующим образом изменяются при разработке новых реализаций. Если при разработке реализации появляются новые АО, они добавляются в раздел нереализованных АО. Спецификации АО определяют имя; количество параметров; имена параметров; описание (комментарии); количество реализаций и их список; признак: является АО корневым (начальным).

Метаалгоритм формируется путем пошагового добавления реализаций АО. Для этого выбирается АО из списков, реализованных или нереализованных, и в спецификации выбирается пункт реализации.

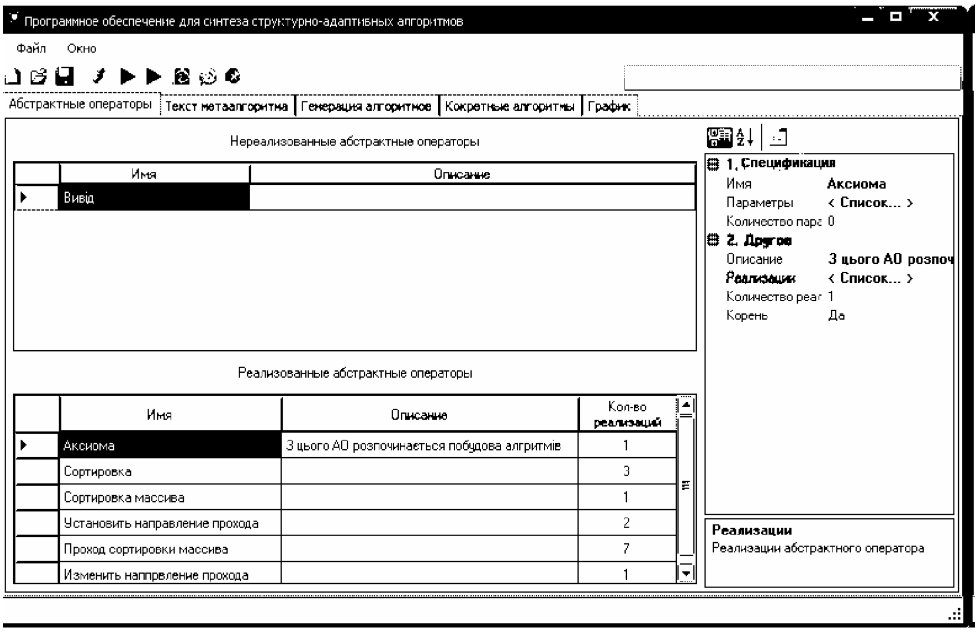


Рисунок 1 – Форма учета и управления АО

В открывшемся окне можно просмотреть список реализаций и их свойства. Свойства включают имя и код реализаций в виде операторов ЯП и других АО.

Редактор АО имеет достаточно полную функциональность. Он позволяет работать с файлами (открыть/сохранить), буфером обмена, имеет средства отмены действий, поиска и замены, визуального выделения синтаксических конструкций.

Выделение конструкций осуществляется согласно описанию ЯП средствами XML. На рис. 2 приведен фрагмент описания ЯП C# и указано выделение препроцессорных переменных (начиная с ##) полужирным начертанием и подчеркиванием.

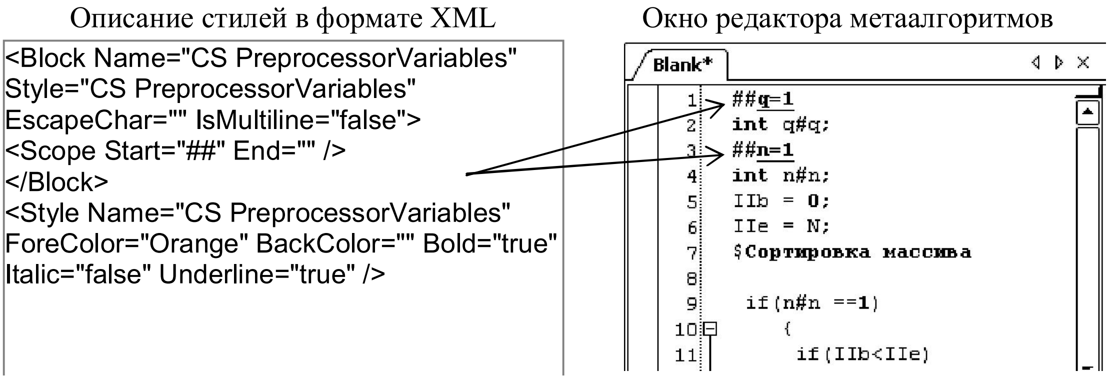


Рисунок 2 – Визуальное выделение синтаксических конструкций

Код реализации (текст программы) нужно вводить на одном из предусмотренных языков программирования.

В реализации АО могут использоваться (вызываться) АО и абстрактные предикаты, как правило, следующего уровня детализации. Для вызова абстрактного оператора используется следующий синтаксис:

$имя АО; имя параметра; … имя параметра.

Вызов абстрактного оператора должен начинаться с начала строки и быть единственным в строке.

После первого вызова абстрактного оператора или предиката он автоматически добавляется в список нереализованных абстрактных операторов. Далее их также нужно реализовать.

Так как АО могут использоваться в конкретных алгоритмах неоднократно, в них могут быть использованы локальные переменные, для организации уникальности имен используется аппарат препроцессорной обработки. Он также необходим для организации множества переменных связующих АО.

Препроцессорная обработка заключается в выполнении препроцессорных операторов присваивания, ветвления и цикла после формирования текста конкретного алгоритма, перед трансляцией.

Возможности препроцессорной обработки представлены в табл. 1.

Таблица 1 – Текст метаалгоритма и результаты его обработки препроцессором

Текст метаалгоритма	##a=3 ##b=5 int a#a; int b#b#a; ##a–1 ##b+2 int a#a int b#a=#b;	##a=3 ##b=5 ##if (#a;#b;<) { int a=3#b; a=#b+#a; } ##else int b=#a;	##a=1 ##b=9 ##for (i;#a;#b;<;2) { int a#i; }
Тот же текст после препроцессорной обработки	int a3; int b53; int a2; int b2=7;	int a=35; a=5+3;	int a1; int a3; int a5; int a7;

Разработанный редактор метаалгоритмов реализует метод пошаговой детализации разработки алгоритмов, заставляет программиста именно так и никак иначе разрабатывать метаалгоритм. В поддержку метода редактор позволяет выполнять навигацию по уровням детализации, вдоль уровня, отслеживать текущий уровень и степень детализации.

Средства отладки метаалгоритмов

В процессе разработки метаалгоритма возможно появление синтаксических и логических ошибок, которые необходимо устранить средствами тестирования и отладки.

Для автоматизации тестирования (а также и процесса адаптации) необходимо использовать транслятор командной строки. Он позволяет построить автоматическую циклическую последовательность:

$$\text{синтез} \rightarrow \text{трансляция} \rightarrow \text{выполнение} \rightarrow \text{проверка}. \qquad (1)$$

На основе метаалгоритма может быть построено бесконечное множество конкретных алгоритмов [2], необходимым этапом является проверка правильности функционирования синтезированных алгоритмов. В большинстве случаев это не является проблемой, и в большинстве случаев проверяющий алгоритм намного проще синтезированного.

Так как любая реализация АО может быть не использована при синтезе конкретных алгоритмов, то ее ошибки могут проявляться не постоянно, а лишь в некоторых из синтезированных алгоритмов. Цикл (1) позволяет выделить те синтезированные конкретные алгоритмы, которые содержат ошибки.

Сложность отладки заключается в том, что ошибки появляются в конкретных алгоритмах, а устранять их нужно в метаалгоритме, т.е. нужно установить обратную связь между конкретным алгоритмом и метаалгоритмом. Для упрощения процесса отладки разработан механизм продвижения сообщений об ошибках от транслятора в командной строке до места ошибки (строки и позиции) в метаалгоритме (рис. 3).



Рисунок 3 – Средства продвижения сообщений об ошибке

Система синтеза при формировании конкретных алгоритмов создает специальный файл, который содержит информацию о порядке включения реализаций АО в конкретный алгоритм. Согласно этой информации строится таблица соответствия строк в конкретном алгоритме и метаалгоритме (рис. 3).

В случае наличия синтаксических ошибок цикл (1) прерывается после трансляции. В редакторе алгоритмов можно просмотреть последний синтезированный алгоритм (с визуальным выделением синтаксических конструкций). Сообщения о синтаксических ошибках из перенаправленного выходного потока транслятора командной строки выводятся на панель редактора. При выделении ошибки соответствующая строка в конкретном алгоритме будет выделена цветом фона, а двойным щелчком открывается окно метаалгоритма с выделением соответствующей строки.

После устранения ошибки цикл (1) возобновляется.

В случае проявления логической ошибки в метаалгоритме цикл (1) прерывается на проверке правильности синтезированного алгоритма. Синтезированный конкретный алгоритм следует отладить традиционными средствами и внести соответствующую правку в метаалгоритм.

Для равномерного покрытия всех операторов метаалгоритма тестирующими выполнениями в цикле (1) на этапе отладки система синтеза равновероятно использует все реализации каждого АО.

Организация процесса адаптации

Последовательность действий при выполнении адаптации можно выразить циклической последовательностью:

синтез → трансляция → выполнение с измерениями → проверка → анализ. (2)

Синтез конкретных алгоритмов заключается в его пошаговом формировании. Начиная с корневого, АО заменяются их реализациями. Выбор конкретных реализаций из числа альтернативных осуществляется на основе рекомендаций подсистемы анализа.

По каждому синтезированному алгоритму в базу данных заносятся два вектора. Первый содержит информацию о структуре синтезированного алгоритма – последовательность реализаций, вошедших в конкретный алгоритм. Он нужен для последующего восстановления лучшего (адаптированного) алгоритма, т.к. тексты всех синтезированных алгоритмов не хранятся. Второй вектор содержит информацию о количестве вхождений каждой реализации в конкретный алгоритм. Он используется системой анализа.

Измерительная система формирует базу знаний в виде:

информация о структуре синтезированного алгоритма →
значение показателя качества.

Анализ эффективности алгоритмов выполняется после накопления достаточной статистической информации (после 50 … 200 выполнений синтезированных алгоритмов) и периодически повторяется при ее значительном пополнении.

Все реализации абстрактных операторов имеют рекомендуемую вероятность их использования в конкретном алгоритме. Задача подсистемы анализа заключается в приведении в соответствие рекомендуемых вероятностей с их эффективностью. Первоначально рекомендуемые вероятности реализаций каждого отдельного АО одинаковы. По результатам анализа рекомендуемая вероятность более полезных реализаций повышается, остальных – понижается.

Посредством рекомендуемых вероятностей использования реализаций АО осуществляется управление синтезом.

Анализ эффективности алгоритмов

Формально задачу нахождения адаптируемого алгоритма можно сформулировать следующим образом. Необходимо найти алгоритм

$$A : t(A) = \min_{B \in \Omega} t(B)\,, \qquad (3)$$

где Ω – множество функционально эквивалентных алгоритмов решения некоторой задачи, которые могут быть построены на основе заданного метаалгоритма; $t(A)$ – значение показателя эффективности алгоритма A на заданном множестве входных данных (время выполнения, степень сжатия данных и т.п.).

Точное решение задачи (3) возможно только методом перебора. Такой метод является неприемлемым из-за большого количества возможных алгоритмов и входных данных. Для решения задачи применен метод направленного случайного поиска. Разработан алгоритм на основе максиминного метода кластеризации [9].

Рассмотрим методику определения рекомендуемых вероятностей.

Рекомендации основываются на допущении, что чем больше используется реализация АО в наиболее эффективном алгоритме, тем она эффективнее и её чаще нужно использовать.

Рекомендуемые вероятности определяются в три этапа. Первый – предварительная обработка, второй – кластеризация, третий – расчет рекомендуемых вероятностей.

На этапе предварительной обработки файла отчета, для каждого r-го выполнения адаптируемого алгоритма из отчета определяется значение показателя эффективности $t(r)$. 50% худших по целевому показателю алгоритмов отбрасывается. Определяется количество включений в каждый r-й выполненный алгоритм каждой k-й реализации АО $Q_{k,r}$ (сквозная нумерация всех реализаций всех АО).

Все $Q_{k,r}$ нормализуются:

$$\overline{Q}_{k,r} = \frac{Q_{k,r} - \min\limits_r Q_{k,r}}{\max\limits_r Q_{k,r} - \min\limits_r Q_{k,r}}. \tag{4}$$

На этапе кластеризации все алгоритмы разбиваются на кластеры – группы, имеющие какие-то свои структурные особенности.

Образ алгоритма в E^n определяется в виде вектора $q(A_r) = [\overline{Q}_{k_1,r}, \overline{Q}_{k_2,r} \dots \overline{Q}_{k_n,r}]$, где n – количество всех реализаций всех АО в метаалгоритме.

Расстояние между образами алгоритмов A_x и A_y определяется как:

$$\rho(A_x, A_y) = \sqrt{\sum_k (\overline{Q}_{k,x} - \overline{Q}_{k,y})^2}. \tag{5}$$

Ищется множество центров кластеров I. Находятся два максимально удаленных друг от друга образа алгоритмов. Они принимаются за центры двух начальных кластеров. Далее последовательно ищутся точки:

$$q(A_z): \min_{A_i \in I}(\rho(A_i, A_z) = \max_{\forall A_j}(\min_{A_i \in I}(\rho(A_i, A_j)).$$

Если $\min\limits_{A_i \in I}(\rho(A_i, A_z) \ge \frac{1}{2} \sum\limits_{A_i, A_j \in I, i \ne j} \rho(A_i, A_j)$, точка $q(A_z)$ принимается за новый центр кластера и алгоритм A_z добавляется к множеству I, в противном случае формирование множества I закончено и остальные точки разносятся к кластерам по критерию минимума расстояния.

На третьем этапе рекомендуемые вероятности определяются следующим образом. Пусть F_i – множество алгоритмов, принадлежащих i-му кластеру, $\check{F}$ – множества алгоритмов, принадлежащих лучшему по среднему значению показателя эффективности кластеру; худший показатель эффективности алгоритма, принадлежащего лучшему кластеру:

$$\check{t}_{\max} = \max_{A_r \in \check{F}}(t_r); \tag{6}$$

лучший показатель эффективности алгоритмов:

$$t_{\min} = \min_{\forall A_r}(t_r). \tag{7}$$

Основываясь на том, что если алгоритм не хуже худшего из лучшего кластера, его можно считать конкурентоспособным, относительное качество алгоритма определим как

$$l_r = \begin{cases} 1 + \dfrac{\check{t}_{\max} - t_r}{\check{t}_{\max} - t_{\min}}, \text{ если } t_r \le \check{t}_{\max} \\ 0, \text{ в противном случае.} \end{cases} \quad (8)$$

Относительное качество алгоритмов в кластере r определим как:

$$P_r = \sum_{A_i \in F_r} l_i . \quad (9)$$

Рекомендуемая вероятность использования кластера как образца для синтеза адаптивного алгоритма:

$$\overline{P}_r = \frac{P_r}{\sum_i P_i} . \quad (10)$$

Если во всех кластерах, кроме лучшего, нет конкурентоспособных алгоритмов, рекомендуемая вероятность лучшего кластера будет равна единице, остальных – нулю.

Рекомендуемая вероятность применения АО $O_{i,j,k}$, при условии выбора z -го кластера, будет:

$$\overline{P}'_k = \frac{P'_k}{\sum_k P'_k}, \text{ где } P'_k = \sum_{A_x \in F_z} (\overline{Q}_{k,x} \cdot l_x) . \quad (11)$$

Заключение

Разработанные методы структурной адаптации алгоритмов и соответствующие программные средства апробированы на алгоритмах сортировки и сжатия данных. Получена сходимость процесса адаптации после 150 … 300 синтезированных алгоритмов.

Апробирована возможность изменения показателя эффективности алгоритмов, в качестве которого использовались время выполнения и степень сжатия данных.

Универсальность разработанных средств обеспечивается описанием синтаксиса ЯП средствами XML. Для изменения ЯП необходимо наличие соответствующего транслятора в командной строке, описание синтаксиса в XML формате и, возможно, преобразователя формата сообщений об ошибках транслятора. Выполнена апробация для ЯП Паскаль и C#.

Разработанные средства позволяют выполнять и альтернативную адаптацию [10], как частный случай структурной.

Правильное применение предложенного метода структурной адаптации алгоритмов и разработанных программных средств позволяет исключить возможность ухудшения показателей эффективности алгоритмов. Однако вопросы, связанные с эффективностью процесса адаптации и его сходимостью, еще не достаточно изучены. Предстоит

исследовать условия сходимости, скорость сходимости и факторы, на нее влияющие, и другие характеристики процесса адаптации.

Литература

1. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео / [Ватолин Д., Ратушняк А., Смирнов М., Юкин В.]. – М. : Диалог-МИФИ, 2002. – 384 с.
2. Шинкаренко В.И. Структурная адаптация алгоритмов на основе полиморфизма / В.И. Шинкаренко // Математические машины и системы. – 2009. – № 2. – С. 28-44.
3. Цейтлин Г.Е. Введение в алгоритмику / Цейтлин Г.Е. – К. : Сфера, 1998. – 310 с.
4. Яценко О.А. Розробка інтегрованих алгебро-алгоритмічних моделей: елементи теорії, інструментарій, застосування : Автореф. дис. на здобуття наук. ступеня канд. фіз.-мат. наук: 01.05.03 / О.А. Яценко / Київський національний ун-т ім. Тараса Шевченка. – К., 2005. – 17 с.
5. Алгеброалгоритмические модели и методы параллельного программирования / [Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.]. – К. : Акакдемпериодика, 2007. – 634 с.
6. Растригин Л.А. Адаптация сложных систем / Растригин Л.А. – Рига : Зинатне, 1981. – 375 с.
7. Шинкаренко В.И. Сравнительный анализ временной эффективности функционально эквивалентных алгоритмов / В.И. Шинкаренко // Проблемы программирования. – 2001. – № 3-4. – С. 31-39.
8. Шинкаренко В.И. Функциональная эффективность нечетко специфицированных алгоритмов / В.И. Шинкаренко // Проблемы программирования. – 2006. – № 1 – С. 24-33.
9. Ту Дж. Принципы распознавания образов / Дж. Ту, Р. Гонсалес – 1978. – 411 с.
10. Шинкаренко В.И. Знание-ориентированный подход к адаптации алгоритмов / В.И. Шинкаренко // Искусственный интеллект. – 2008. – № 3. – С. 388-397.

В.І. Шинкаренко, Г.Г. Кроль, І.В. Литвин, Є.Г. Васецький

Методи та засоби структурної адаптації алгоритмів на метаалгоритмічній основі

Розглядається задача структурної адаптації алгоритмів. Представлені достатньо універсальні засоби адаптації алгоритмів у складі різноманітного прикладного програмного забезпечення. Розроблені повнофункціональний редактор та спеціалізовані засоби відлагодження метаалгоритмів, підсистема синтезу адаптивних алгоритмів. Підсистема аналізу ефективності алгоритмів, що базується на кластеризації методом максимінної відстані, виробляє базу знань і тим самим керує процесами синтезу та адаптації.

V.I. Shynkarenko, G.G. Krol, I.V. Litvin, Ye.G. Vasetsky

Methods and Tools for Structural Adaptation of Algorithms Based on a Metaalgorithm

The problem of structural adaptation of algorithms is considered. Universal tools for adaptation of algorithms as a part of the various applied software are presented. The full-function editor and specialised debugger of metaalgorithms, a subsystem of synthesis of adaptive algorithms are developed. The subsystem of the analysis of algorithms efficiency works out the knowledge base and by that processes the synthesis and adaptation control. The subsystem of the analysis grounded on clustering by a method maxmin distances.

Статья поступила в редакцию 22.06.2009.